

09/093,533

EM302071688US

CASH-001

This application is submitted in the name of the following inventors:

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21

<u>Inventor</u>	<u>Citizenship</u>	<u>Residence Address</u>
Malcolm, Michael	United States	250 Family Farm Road Woodside, California United States
Zarnke, Robert	Canada	383 Parkgreen Place Waterloo, Ontario N2L 5S6 Canada

The assignee is CacheFlow Inc., a Washington corporation having an office at 8653 - 154th Avenue, Redmond WA 98052.

Title of the Invention

Network Object Cache Engine

///

Background of the Invention1. *Field of the Invention*

This invention relates to devices for caching objects transmitted using a computer network.

2. *Related Art*

In computer networks for transmitting information, information providers (sometimes called "servers") are often called upon to transmit the same or similar information to multiple recipients (sometimes called "clients") or to the same recipient multiple times. This can result in transmitting the same or similar information multiple times, which can tax the communication structure of the network and the resources of the server, and cause clients to suffer from relatively long response times. This problem is especially acute in several situations: (a) where a particular server is, or suddenly becomes, relatively popular; (b) where the information from a particular server is routinely distributed to a relatively large number of clients; (c) where the information from the particular server is relatively time-critical; and (d) where the communication path between the server and its clients, or between the clients and the network, is relatively slow.

1 One known method is to provide a device (such as a general purpose
2 processor operating under software control) which acts as a proxy, receiving requests
3 for information from one or more clients, obtaining that information from one or more
4 servers, and transmitting that information to the clients in place of the servers. When the
5 proxy has previously obtained the information from one or more servers, it can deliver
6 that information to the client without having to repeat the request to the server. While
7 this method achieves the goal of reducing traffic in the network and load on the server,
8 it has the drawback that significant overhead is required by the local operating system
9 and the local file system or file server of the proxy. This adds to the expense of
10 operating the network and slows down the communication path between the server and
11 the client.

12 There are several sources of delay, caused primarily by the proxy's
13 surrendering control of its storage to its local operating system and local file system: (a)
14 the proxy is unable to organize the information from the server in its mass storage for
15 most rapid access; and (b) the proxy is unable to delete old network objects received
16 from the servers and store new network objects received from the servers in a manner
17 which optimizes access to mass storage. In addition to the added expense and delay, the
18 proxy's surrendering control of its storage restricts functionality of the proxy's use of its
19 storage: (a) it is difficult or impossible to add to or subtract from storage allocated to the
20 proxy while the proxy is operating; and (b) the proxy and its local file system cannot
21 recover from loss of any part of its storage without using an expensive redundant
22 storage technique, such as a RAID storage system.
23

Summary of the Invention

4

1 In a preferred embodiment, the cache engine collects information to be
2 written to disk in write episodes, so as to maximize efficiency when writing information
3 to disk and so as to maximize efficiency when later reading that information from disk.
4 The cache engine performs write episodes so as to atomically commit changes to disk
5 during each write episode, so the cache engine does not fail in response to loss of power
6 or storage, or other intermediate failure of portions of the cache. The cache engine
7 stores key system objects on each one of a plurality of disks, so as to maintain the cache
8 holographic in the sense that loss of any subset of the disks merely decreases the
9 amount of available cache. The cache engine selects information to be deleted from disk
10 in delete episodes, so as to maximize efficiency when deleting information from disk and
11 so as to maximize efficiency when later writing new information to those areas of disk.
12 The cache engine responds to the addition or deletion of disks as the expansion or
13 contraction of the amount of available cache.
14

15 Brief Description of the Drawings

16
17 Figure 1 shows a block diagram of a network object cache engine in a
18 computer network.
19

20 Figure 2 shows a block diagram of a data structure for maintaining storage
21 blocks for a set of cached network objects.
22

1 Figure 3 shows a block diagram of data structures for caching network
2 objects.

3
4 Figure 4 shows a block diagram of a set of original and modified blocks.

5
6 Figure 5 shows a flow diagram of a method for atomic writing of modified
7 blocks to a single disk drive.

8
9 Figure 6 shows a block diagram of a set of pointers and regions on mass
10 storage.

11 Detailed Description of the Preferred Embodiment

12
13
14 In the following description, a preferred embodiment of the invention is
15 described with regard to preferred process steps and data structures. Those skilled in the
16 art would recognize after perusal of this application that embodiments of the invention
17 can be implemented using general purpose processors and storage devices, special
18 purpose processors and storage devices, or other circuits adapted to particular process
19 steps and data structures described herein, and that implementation of the process steps
20 and data structures described herein would not require undue experimentation or further
21 invention.

22
23 / / /

1 *Caching Network Objects*

2
3 Figure 1 shows a block diagram of a network object cache engine in a
4 computer network.

5
6 A cache engine 100 is coupled to a computer network 110, so that the
7 cache engine 100 can receive messages from a set of devices 111 also coupled to the
8 network 110.

9
10 In a preferred embodiment, the network 110 includes a plurality of such
11 devices 111, interconnected using a communication medium 112. For example, where
12 the network 110 includes a LAN (local area network), the communication medium 112
13 may comprise ethernet cabling, fiber optic coupling, or other media. The network 110
14 preferably includes a network of networks, sometimes called an "internet" or an
15 "intranet."
16

17 In a preferred embodiment, the devices 111 coupled to the network 110
18 communicate with the cache engine 100 using one or more protocols for communication,
19 such as HTTP (hypertext transfer protocol) or one of its variants, FTP (file transfer
20 protocol), or other protocols.

21
22 The cache engine 100 includes a processor 101 and a cache 102. In a
23 preferred embodiment, the processor 101 comprises a general purpose processor

1 operating under software control to perform the methods described herein and to
2 construct and use the data structures described herein; as used herein, when the cache
3 engine 100 performs particular tasks or maintains particular data structures that reference
4 includes condign operation by the processor 101 under control of software maintained
5 in a program and data memory 103.

6
7 The cache 102 includes the program and data memory 103 and a mass
8 storage 104. In a preferred embodiment, the mass storage 104 includes a plurality of disk
9 drives such as magnetic disk drives, but may alternatively include optical or magneto-
10 optical disk drives. As used herein, references to "disk" and "disk drives" refer to the
11 mass storage 104 and its individual drives, even if the mass storage 104 and its individual
12 drives do not include physical disk-shaped elements. The cache engine 100 is coupled
13 to the network 110 and can receive and transmit a set of protocol messages 113
14 according to the one or more protocols with which the devices 111 communicate with
15 the cache engine 100.
16

17 The cache engine 100 maintains a set of network objects 114 in the cache
18 102. The cache engine 100 receives protocol messages 113 from a set of "client"
19 devices 111 to request network objects 114 to be retrieved from a set of "server"
20 devices 111. In response thereto, the cache engine 100 issues protocol messages 113 to
21 request those network objects 114 from one or more server devices 111, receives those
22 network objects 114 and stores them in the cache 102, and transmits those network
23 objects 114 to the requesting client devices 111.

1 As used herein, the terms "client" and "server" refer to a relationship
2 between the client or server and the cache engine 100, not necessarily to particular
3 physical devices 111. As used herein, one "client device" 111 or one "server device"
4 111 can comprise any of the following: (a) a single physical device 111 executing
5 software which bears a client or server relationship to the cache engine 100; (b) a
6 portion of a physical device 111, such as a software process or set of software processes
7 executing on one hardware device 111, which portion of the physical device 111 bears a
8 client or server relationship to the cache engine 100; or (c) a plurality of physical devices
9 111, or portions thereof, cooperating to form a logical entity which bears a client or
10 server relationship to the cache engine 100. The phrases "client device" and "server
11 device" refer to such logical entities and not necessarily to particular individual physical
12 devices 111.

13
14 The cache engine 100 preserves the network objects 114 in the cache 102,
15 and reuses those network objects 114 by continuing to serve them to client devices 111
16 which request them. When the cache 102 becomes sufficiently full, the cache engine
17 100 removes network objects 114 from the cache 102. For example, the cache engine
18 100 can remove objects as described herein in the section "Removing Objects from
19 Cache."

20
21 In a preferred embodiment, the cache engine 100 uses the memory 103 as a
22 cache for those network objects 114 maintained using the mass storage 104, while using

1 the combined memory 103 and mass storage 104 as the cache 102 for those network
2 objects 114 available on the network 110.

3
4 The cache 102 is not a file storage system, and network objects 114 which
5 are stored in the cache 102 may be removed automatically from the cache 102 at any
6 time by the cache engine 100. All network objects 114 and all other data maintained by
7 the cache 102 is transient, except for a very small number of system objects which are
8 required for operation, and those system objects are redundantly maintained on the mass
9 storage 104 so as preserve those system objects against possible loss of a part of the
10 mass storage 104 (such as loss of one or more disk drives). Thus the cache engine 100
11 need not guarantee that network objects 114 which are stored in the cache 102 will be
12 available at any particular time after they are stored, and failure or even intentional
13 removal of portions of the cache 102 (such as portions of the mass storage 104) cannot
14 cause failure of the cache engine 100. Similarly, recovery or intentional addition of
15 additional mass storage 104 (such as "hot swapping" of disk drives) is smoothly
16 integrated into the cache 102 without interruption of operation of the cache engine 100.

17
18 Moreover, the cache engine 100 operates exclusively to perform the
19 operation of caching the network objects 114. There is no separate "operating system,"
20 no user, and there are no user application programs which execute independently on the
21 processor 101. Within the memory 103, there are no separate memory spaces for "user"
22 and "operating system." The cache engine 100 itself maintains the cache 102 of the
23 network objects 114 and selects the network objects 114 for retention in the cache 102

1 or removal from the cache 102, operating so as to (1) localize writing the network
2 objects 114 to the mass storage 104, (2) localize deletion of the network objects 114 from
3 the mass storage 104, and (3) efficiently replace the network objects 114 in the cache
4 102 with new network objects 114. In a preferred embodiment, the cache engine 100
5 performs these operations efficiently while operating the cache 102 relatively filled with
6 network objects 114.

7
8 In a preferred embodiment, the cache engine 100 maintains statistics
9 regarding access to the cache 102. These statistics can include the following:

- 10
11 o a set of hit rates for the cache 102, including (1) a hit rate for network objects 114
12 found in the cache 102 versus those which must be retrieved from server devices
13 111, and (2) a hit rate for network objects 114 found in the memory 103 versus
14 those which must be retrieved from the mass storage 104;
- 15
16 o a set of statistics for operations on the memory 103, including (1) the number of
17 network objects 114 which are maintained in the memory 103, and (2) the fraction
18 of memory 103 which is devoted to caching network objects 114 versus storing
19 system objects or unallocated; and
- 20
21 o a set of statistics for operations on the mass storage 104, including (1) the number
22 of read operations from the mass storage 104, (2) the number of write operations
23 to the mass storage 104, including the number of "write episodes" as described

1 herein, and (3) the fraction of the mass storage 104 which is devoted to caching
2 network objects 114 versus storing system objects or unallocated.

3
4 The cache engine 100 can also maintain statistics which are combinations
5 or variants of the above.

6
7 *Using the Cache Engine*

8
9 There are numerous circumstances in which the cache engine 100 can
10 provide improved performance or additional functionality in the network 110. For
11 example, the cache engine 100 can be used as a proxy cache (whether to provide a
12 firewall, to provide a cache for client devices 111 coupled to a local area network, or
13 otherwise), as a reverse proxy cache, as a cache for requests made by users of a single
14 ISP, as a cache for "push" protocols, or as an accelerator or server cache.

15
16 The cache engine 100 provides the client devices 111 with relatively
17 quicker access to network objects 114 otherwise available directly from the server
18 devices 111. Typically the client devices 111 request those network objects 114 from the
19 cache engine 100, which either transmits them to the client devices 111 from the cache
20 102 or obtains them from the server devices 111 and then transmits them to the client
21 devices 111.

1 The cache engine 100 can exercise more intelligence and proactivity than
2 simply waiting for documents to be requested by the client devices 111:

3
4 o The cache engine 100 can be configured preloaded with selected network
5 objects 114 which are expected to be requested by the client devices 111. For
6 example, certain network objects 114 are known to be commonly requested by
7 client devices 111 throughout the network 110 known as the internet; these
8 network objects 114 can be preloaded in the cache engine 100 upon
9 manufacture. These network objects 114 could include home pages for well-
10 known companies (such as Netscape) and well-known search engines (such as
11 Digital's "Alta Vista").

12
13 o The cache engine 100 can periodically request network objects 114 responsive to
14 a set of statistics regarding commonly requested network objects 114. For
15 example, information regarding commonly requested network objects 114 can be
16 maintained on a server device 111; the cache engine 100 can request this
17 information from the server device 111 and periodically request those network
18 objects 114 for storage in the cache 102. In a preferred embodiment, the cache
19 engine 100 can perform this operation periodically when client devices 111 are
20 not actively using the cache engine 100, such as relatively unloaded times in the
21 late night or early morning.

22

1 o The cache engine 100 can periodically request network objects 114 responsive to
2 a set of user preferences at the client devices 111. For example, the cache engine
3 100 can receive (either upon request or otherwise) a set of bookmarks from the
4 client devices 111 and can request those network objects 114 from the server
5 devices 111. In a preferred embodiment, the cache engine 100 can request those
6 network objects 114 which have changed in a selected time period such as one
7 day.

8
9 o The cache engine 100 can provide a mirror site to one or more server devices 111,
10 by periodically, or upon request, receiving network objects 114 from the server
11 devices 111 to be delivered by the server device 111 to client devices 111 which
12 have changed in a selected time period such as one day.

13
14 o The cache engine 100 can provide an accelerator for one or more server devices
15 111, by receiving requests to the server devices 111 which are distributed among a
16 plurality of cache engines 100. Each cache engine 100 maintains its cache 102
17 with network objects 114 to be delivered by the server device 111 to client
18 devices 111. Service by the server device 111 is thus accelerated, because each
19 cache engine 100 can respond to some of the load of requests for information,
20 while limiting the number of requests for information which are passed through
21 and must be handled by the server device 111 itself.

1 o The cache engine 100 can provide a first type of push protocol assist to one or
2 more server devices 111, by transmitting network objects 114 to one or more
3 client devices 111 or proxy caches using a push protocol. For example, when the
4 server devices 111 provide a network broadcast service, the cache engine 100 can
5 receive network objects 114 from the server devices 111 to be broadcast to a
6 subset of the network 110 and can independently broadcast those network
7 objects 114.

8
9 o The cache engine 100 can provide a second type of push protocol assist to one or
10 more server devices 111, by allowing those server devices 111 to broadcast
11 network objects 114 to a plurality of cache engines 100. Each cache engine 100
12 can make the broadcast network objects 114 available to client devices 111 which
13 request those network objects 114 from the cache engine 100 as if the cache
14 engine 100 were the server device 111 for those network objects 114.

15
16 The network objects 114 can include data, such as HTML pages, text,
17 graphics, photographs, audio, video; programs, such as Java or ActiveX applets or
18 applications; or other types of network objects, such as push protocol objects. The
19 cache engine 100 can record frames of streaming audio or streaming video information in
20 the cache 102, for delayed use by a plurality of client devices 111. Some types of known
21 network objects 114 are not cached, such as CGI output or items marked noncacheable
22 by the server device 111.

1 In a preferred embodiment, the cache engine 100 can glean knowledge
2 about the client devices 111 from the protocol messages 113 or by other means, such as
3 interrogating routing devices in the network 110, and can react in response to that
4 information to provide differing network objects 114 to differing client devices 111. For
5 example, the cache engine 100 can select server devices 111 for proximity or content in
6 response to information about client devices 111, as follows:

7
8 o The cache engine 100 can select a particular server device 111 for rapid response,
9 such as for network routing proximity or for spreading service load over a
10 plurality of server devices 111.

11
12 o The cache engine 100 can select content at the server device 111 in response to
13 information about the client device 111, such as tailoring the language of the
14 response (such as serving pages in the English language or the French language),
15 or such as tailoring local information (such as advertising, news, or weather). In a
16 preferred embodiment, local information such as advertising can be retrieved from
17 a local server device 111 which supplies advertising for insertion into pages to be
18 served to local client devices 111.

19
20 *The Cache*

21
22 Figure 2 shows a block diagram of a data structure for maintaining storage
23 blocks for a set of cached network objects.

1 The cache 102 includes a set of blocks 200, each of which comprises 4096
2 bytes in a preferred embodiment, and each of which can be stored in the memory 103 or
3 on the mass storage 104. In alternative embodiments, each of the blocks 200 can
4 comprise a size other than 4096 bytes, and may be responsive to an amount of available
5 memory 103 or mass storage 104.

6
7 Each of the blocks 200 can comprise either a data block 200, which
8 includes data, that is, information not used by the cache engine 100 but maintained for
9 the client devices 111, or control information, that is, information used by the cache
10 engine 100 and not used by the client devices 111.

11
12 The blocks 200 are organized into a set of objects 210, each of which
13 comprises an object descriptor 211, a set of data blocks 200, and a set of block pointers
14 212 referencing the data blocks 200 from the object descriptor 211. The object
15 descriptor 211 comprises a separate control block 200. Where the block pointers 212
16 will not fit into a single control block 200, or for other types of relatively larger objects
17 210, the object descriptor 211 can reference a set of indirect blocks 216, each of which
18 references inferior indirect blocks 216 or data blocks 200. Each indirect block 216
19 comprises a separate control block 200. Relatively smaller objects 210 do not require
20 indirect blocks 216.

21

1 The block pointers 212 each comprise a pointer value 215 comprising a
2 single 32-bit word and indicating the location of the block 200 on the mass storage 104,
3 such as a physical disk block address.

4
5 In an alternative embodiment, the block pointers 212 each comprise a first
6 bit 213 indicating whether the referenced block 200 is stored in the memory 103 or the
7 mass storage 104, a second bit 214 indicating whether the referenced block 200 is a
8 control block 200 (comprising control information) or a data block 200 (comprising data
9 for network objects 114), and the pointer value 215 comprises a 30-bit value indicating
10 the location of the block 200. In such alternative embodiments, when the block 200 is
11 stored in the memory 103, the pointer value 215 indicates a byte address in the memory
12 103; when the block is stored on the mass storage 104, the pointer value 215 indicates a
13 physical disk block address on the mass storage 104.

14
15 In a preferred embodiment, the objects 210 are each referenced by a root
16 object 220, which is maintained redundantly in a plurality of (preferably two) copies of a
17 root block 221 on each disk drive of the mass storage 104. In a preferred embodiment,
18 there is one root object 220 for each disk drive of the mass storage 104. Thus, each disk
19 drive of the mass storage 104 has a separate root object 210, which is maintained using
20 two copies of its root block 221. Each disk drive's root object 220 references each
21 current object 210 for that disk drive.

1 In a preferred embodiment, one copy of the root block 221 is maintained in
2 each of physical disk blocks 2 and 3 of each of the disk drives of the mass storage 104.
3 When the root block 221 for that disk drive is written to the mass storage 104, it is first
4 written to the physical disk block 2, and then identically written to the physical disk
5 block 3. When the cache engine 100 is started or restarted, the root block 221 is read
6 from the physical disk block 2. If this read operation is successful, it is then identically
7 rewritten to the physical disk block 3; however, if this read operation is unsuccessful, the
8 root block 221 is instead read from the physical disk block 3, and then identically
9 rewritten to the physical disk block 2.

10
11 In a preferred embodiment, the cache engine 100 also stores certain system
12 objects 210 redundantly on each disk drive on the mass storage 104, so as to maintain
13 the cache 102 holographic in the sense that loss of any subset of the disk drives merely
14 decreases the amount of available cache. Thus, each such system object 210 is
15 referenced by the root object 220 for its disk drive and is maintained using two copies of
16 its object descriptor 211. These system objects 210 which are maintained redundantly
17 include the root object 220, a blockmap object 210, and a hash table 350 (figure 3), each
18 as described herein, as well as other system objects, such as objects 210 for collected
19 statistics, documentation, and program code.

20
21 A subset of the blocks 200 are maintained in the memory 103, so as to use
22 the memory 103 as a cache for the mass storage 104 (just as the memory 103 and the
23 mass storage 104 collectively act as the cache 102 for network objects 114). The blocks

1 200 maintained in the memory 103 are referenced by a set of block handles 230, which
2 are also maintained in the memory 103.
3

4 Each of the block handles 230 includes a forward handle pointer 232, a
5 backward handle pointer 233, a reference counter 234, a block address 235, a buffer
6 pointer 236, and a set of flags 237.
7

8 The forward handle pointer 232 and the backward handle pointer 233
9 reference other block handles 230 in a doubly-linked list of block handles 230.
10

11 The reference counter 234 maintains a count of references to the block
12 200 by processes of the cache engine 100. The reference counter 234 is updated when
13 a block handle 230 for the block 200 is claimed or released by a process for the cache
14 engine 100. When the reference counter 234 reaches zero, there are no references to the
15 block 200, and it is placed on a free list of available blocks 200 after having been written
16 to disk, if it has been modified, in the next write episode.
17

18 The block address 235 has the same format as the block pointer 212. The
19 buffer pointer 236 references a buffer used for the block 200. The flags 237 record
20 additional information about the block 200.
21

22 In one embodiment, the block handles 230 are also threaded using a set of
23 2Q pointers 238 and a 2Q reference counter 239, using the "2Q" technique, as further

1 described in "2Q: A Low Overhead High Performance Buffer Management
2 Replacement Algorithm," by Theodore Johnson and Dennis Shasha, hereby
3 incorporated by reference as if fully set forth herein.

4

5 *How Network Objects are Cached*

6

7 Figure 3 shows a block diagram of data structures for caching network
8 objects.

9

10 The cache engine 100 receives protocol requests from the network 110. In
11 a preferred embodiment, each protocol request uses the HTTP protocol (or a variant such
12 as SHTTP), and each HTTP request includes a URL (uniform resource locator) 310,
13 which identifies a network object 114 in the network 110. In a preferred embodiment,
14 each URL 310 identifies the server device 111 for the network object 114 and the
15 location of the network object 114 on that server device 111.

16

17 In alternative embodiments, the cache engine 100 may use other protocols
18 besides HTTP or its variants, and the cache engine 100 may be responsive to one or more
19 other identifiers for network objects 114 besides its URL 310. Accordingly, as used
20 herein, the term "URL" refers generally to any type of identifier which is capable of
21 identifying, or assisting in identifying, a particular network object 114.

22

1 The URL 310 includes a host identifier, which identifies the server device
2 111 at which the network object 114 is located, and a document identifier, which
3 identifies the location at which the network object 114 is located at the server device
4 111. In a preferred embodiment, the host identifier comprises a character string name for
5 the server device 111, which can be resolved to an IP (internet protocol) address.
6 However, in alternative embodiments, the host identifier may comprise the IP address for
7 the server device 111, rather than the character string name for the server device 111.

8
9 The cache engine 100 includes a hash function 320 which associates the
10 URL 310 with a hash signature 330, which indexes a hash bucket 340 in a hash table
11 350 in the cache 102. In a preferred embodiment, the hash table 350 comprises a set of
12 hash tables 350, one for each disk drive, each of which references those network objects
13 114 which are stored in the cache 102 on that disk drive of the mass storage 104. Each
14 such hash table 350 has its own object descriptor 211; collectively the hash tables 350
15 form a single logical hash table.

16
17 In a preferred embodiment, the hash signature 330 comprises a 32-bit
18 unsigned integer value which is determined responsive to the URL 310, and which is
19 expected to be relatively uniformly distributed over the range of all possible 32-bit
20 unsigned integer values. In a preferred embodiment, the URL 310 is also associated with
21 a 64-bit URL signature which is also an unsigned integer value, determined responsive
22 to the URL 310, and which is expected to be relatively uniformly distributed over the
23 range of all possible 64-bit unsigned integer values; when comparing URLs 310, the

1 URL signatures are compared first, and only if they are equal are the URLs 310
2 themselves compared. In a preferred embodiment, the URL 310 is also converted to a
3 canonical form prior to determining the hash signature 330 or the URL signature, such
4 as by converting all alphabetic characters therein into a single case (lower case or upper
5 case). In a preferred embodiment, each non-null hash bucket 340 comprises one data
6 block 200.

7
8 Because the hash table 350 associates the URL 310 directly with the hash
9 bucket 340 in the hash table 350, storage of the network objects 114 in the cache 102 is
10 not hierarchical; each of the network objects 114 can be referenced and accessed from
11 the cache 102 within order of constant time, such as less than about two disk read
12 access times. Moreover, there is no special requirement that the network objects 114 in
13 the cache 102 must have unique names; when network objects 114 have identical names
14 (such as when they are old and new versions of the same network object 114), the hash
15 table 350 simply points to the same hash bucket 340 for both of them.

16
17 When there are both old and new versions of the same network object 114,
18 the cache engine 100 resolves new references by the URL 310 only to the new version
19 of the network object 114. Those client devices 111 which are already accessing the old
20 version of the network object 114 when the new version of the network object 114 is
21 stored in the cache 102 will continue to access the old version of the network object
22 114. However, subsequent accesses to that network object 114, even by the same client
23 device 111, using the URL 310 will be resolved by the cache engine 100 to the new

1 version of the network object 114. The old version of the network object 114 is deleted
2 as soon as possible when all client devices 111 are done using it.
3

4 The cache 102 differs from a file system also in that the client device 111
5 has no control over storage of the network objects 114 in the cache 102, including (1)
6 the name space at the cache 102 for storage of the network objects 114, (2) the ability to
7 name or rename the network objects 114, (3) whether the network objects 114 are
8 removed from the cache 102 at any time, and (4) whether the network objects 114 are
9 even stored in the cache 102 at all.

10
11 In a preferred embodiment, the cache engine 100 uses the memory 103 and
12 the mass storage 104 (preferably a plurality of magnetic disk drives) to cache the
13 network objects 114 so as to maintain in the cache 102 those network objects 114 most
14 likely to be required by the client device 111. However, in alternative embodiments, the
15 cache engine 100 may enforce selected administrative requirements in addition to
16 maintaining network objects 114 most likely to be used by the client device 111, such as
17 preferring or proscribing certain classes of network objects 114 or certain classes of
18 client devices 111 or server devices 111, whether at all times or at selected times of day
19 and selected days.

20

21 The cache engine 100 uses the hash function 320 and the hash table 350
22 to identify an object 210 (and thus one or more data blocks 200) associated with the
23 URL 310 (and thus associated with the network object 114). The cache engine 100

1 operates on the object 210 to retrieve from the cache 102 the network object 114
2 requested by the HTTP request, and to deliver that network object 114 to the client
3 device 111. The cache engine 100 maintains the cache 102 using the memory 103 and
4 the mass storage 104 so that whether the object 210 is in the cache 102, and if in the
5 cache 102, whether the object 210 is in the memory 103 or on the mass storage 104 is
6 transparent to the client device 111 (except possibly for different time delays in
7 retrieving the object 210 from the memory 103 or from the mass storage 104).

8
9 As described herein in the section "Writing to Disk," the cache engine 100
10 writes blocks 200 (and objects 210 comprising those blocks 200) from the memory 103
11 to the mass storage 104 on occasion, so as to maintain those blocks 200 in the memory
12 103 which are most frequently accessed.

13
14 As described herein, when writing blocks 200 from the memory 103 to the
15 mass storage 104, the cache engine 100 controls where the blocks 200 are written onto
16 the mass storage 104 (such as determining onto which disk drive for the mass storage
17 104 and which location on that disk drive), and when the blocks 200 are written onto
18 the mass storage 104 (such as determining at which times it is advantageous to write
19 data onto the mass storage 104). The cache engine 100 attempts to optimize the times
20 and locations when and where the blocks 200 are written to disk, so as to minimize time
21 and space required to write to and read from disk.

22

1 The hash table 350 is a system object 210, and similar to other system
2 objects 210, includes an object descriptor 211, zero or more indirect blocks 216, and zero
3 or more data blocks 200. Because the hash table 350 is expected to be used relatively
4 frequently, its indirect blocks 216 are expected to all be maintained in the memory 103,
5 although for a relatively large hash table 350 some of its data blocks 200 will be
6 maintained on the mass storage 104. In a preferred embodiment, the hash table 350 is
7 distributed over the plurality of disk drives for the mass storage 104, and the portion of
8 the hash table 350 for each disk drive is referenced in the root object 220 for that disk
9 drive.

10
11 Each hash signature 330 is indexed into the hash table 350 using the hash
12 signature 330 modulo the number of hash buckets 340 in the hash table 350. Each hash
13 bucket 340 comprises one block 200. Each hash bucket 340 includes zero or more hash
14 entries 360; each hash entry 360 includes a reference to the object 210 at the hash entry
15 360 (comprising a pointer to the object descriptor 211 for that object 210).

16
17 The hash bucket 340 includes a secondary hash table, having a plurality of
18 chains of secondary hash table entries (such as, for example, 32 such chains). The hash
19 signature 330 is used to select one of the chains so as to search for the hash entry 360
20 associated with the URL 310.

21
22 In an alternative embodiment, the hash entries 360 are maintained within
23 the hash bucket 340 in an ordered list by a secondary hash value, with null entries

1 possibly interspersed (when the associated network objects 114 have been deleted or
2 otherwise removed from the hash table 350); the secondary hash value is also
3 determined in response to the hash signature 330, such as by computing the hash
4 signature 330 modulo a selected value such as $2^{**}32$. If there are multiple hash entries
5 360 with the same secondary hash value, the cache engine 100 examines the object
6 descriptor 211 associated with each one of the multiple hash entries 360 for the URL
7 310 of the correct network object 114 associated with the URL 310 having the
8 associated hash signature 330.

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

In a preferred embodiment, each hash bucket 340 has a selected size which
is sufficient to hold at least 1.5 to 2 times the number of expected hash entries 360 if the
hash entries 360 were perfectly uniformly distributed (this selected size is preferably
exactly one data block 200). If a hash entry 360 is assigned to a hash bucket 340
which is full, one of the network objects 114 already associated with the hash bucket
340, along with its associated hash entry 360, is deleted from the hash bucket 340 and
from the cache 102 to make room for the new hash entry 360.

In a preferred embodiment, there can be a plurality of different operational
policies for selecting just which objects 210 are deletable.

/ / /

Mass Storage with Multiple Disk Drives

The cache engine 100 maintains a DSD (disk set descriptor) object 210 for each disk drive currently or recently present on the mass storage 104, which includes a data structure describing that disk drive. The cache engine 100 also maintains a DS (disk set) object 210, which references all of the DSD objects 210, and which is maintained redundantly on one or more of the disk drives for the mass storage 104. Thus, the DS object 210 is maintained redundant on the mass storage 104 on a plurality of disk drives (preferably all of them), with each disk drive's information being maintained on that disk drive in the DSD object 210.

Each DSD object 210 includes at least the following information: (1) the number of disk drives; (2) the collective total size of all disk drives; (3) for each disk drive--the individual size of that disk drive, an identifier for that disk drive, and a index into an array of all the disk drives; and (4) for each disk drive---the range of hash signatures 330 which are maintained on that disk drive. Also, the range of hash signatures 330 which are maintained on each disk drive is maintained in a separate system object 210 which maps each hash signature 330 to a particular disk drive. In a preferred embodiment, sizes are expressed as multiples of a selected value such as 1 megabyte.

1 The hash entries 360 are distributed over the plurality of disk drives in
2 proportion to the size of each disk drive, rounded to an integer number of hash entries
3 360.

4
5 When a disk drive is added, removed, or replaced, the cache engine 100
6 creates or modifies an associated DSD object 210, and updates the DS object 210. This
7 operation proceeds in like manner as updating a data block 200; thus, any control
8 blocks 200 which reference the DS object 210 or one of the DSD objects 210 are also
9 updated, and the update is atomically committed to the mass storage 104 with the next
10 write episode. (Updates to the DS object 210 are atomically committed for each disk
11 drive, one at a time.) Thus, the mass storage 104 can be dynamically updated, including
12 changing the identity or number of disk drives, while the cache engine 100 continues to
13 operate, and the only effect on the cache engine 100 is to alter its perception of the
14 amount of mass storage 104 which is available for the cache 102.

15 16 *Writing to Disk*

17
18 The cache engine 100 implements a "delayed write" technique, in which
19 the objects 210 which are written into the cache 102 (including objects 210 which are
20 new versions of old objects 210 already present in the cache 102) are written first into
21 the memory 103, and only later written out to the mass storage 104. Unlike file systems
22 which use delayed write techniques, there is no need to provide a non-volatile RAM or
23 a UPS (uninterruptable power supply) and an associated orderly shutdown procedure,

1 because the cache engine 100 makes no guarantee of persistence for the network
2 objects 114 in the cache 102. For example, if a particular network object 114 is lost from
3 the cache 102, that network object 114 can typically be reacquired from its associated
4 server device 111.

5
6 However, the delayed write technique operates to maintain consistency of
7 the cache 102, by not overwriting either control blocks 200 or data blocks 200 (except
8 for the root block 221). Instead, modified blocks 200 are written to the mass storage
9 104, substituted for the original blocks 200, and the original blocks 200 are freed, all in
10 an atomic operation called a "write episode." If a write episode is interrupted or
11 otherwise fails, the entire write episode fails atomically and the original blocks 200
12 remain valid.

13
14 A modified data block 200 is created when the underlying data for the
15 original data block 200 is modified (or when new underlying data, such as for a new
16 network object 114, is stored in a new data block 200). A modified control block 200 is
17 created when one of the original blocks 200 (original data block 200 or original control
18 block 200) referenced by the original control block 200 is replaced with a modified
19 block 200 (modified data block 200, new data block 200, or modified control block
20 200); the modified control block 200 references the modified block 200 rather than the
21 original block 200.

22

Each write episode is structured so as to optimize both the operation of writing blocks 200 to the mass storage 104 and later operations of reading those blocks 200 from the mass storage 104. The following techniques are used to achieve the read and write optimization goals:

o modified blocks 200 to be written are collected and written, when possible, into sequential tracks of one of the disk drives used for the mass storage 104;

o indirect blocks 216 are written to storage blocks which are close to and before those data blocks 200 which they reference, so as to enable reading the referenced data blocks 200 in the same read operation whenever possible;

o sequentially related data blocks 200 are written to sequential free storage blocks (if possible, contiguous free storage blocks) on one of the disk drives used for the mass storage 104, so as to enable reading the related data blocks 200 in the same read operation whenever possible;

o blocks 200 (control blocks 200 or data blocks 200) to be written are collected together for their associated objects 210 and ordered within each object 210 by relative address, so as to enable reading blocks 200 for a particular object 210 in the same read operation whenever possible.

Figure 4 shows a block diagram of a set of original and modified blocks.

1 Figure 5 shows a flow diagram of a method for atomic writing of modified
2 blocks to a single disk drive.

3
4 A tree structure 400 (figure 4) of blocks 200 includes the original control
5 blocks 200 and the original data blocks 200, which have been already written to the
6 mass storage 104 and referenced by the root object 220. Some or all of these original
7 blocks 200 can be held in the memory 103 for use.

8
9 A method 500 (figure 5) includes a set of flow points to be noted, and
10 steps to be executed, by the cache engine 100.

11
12 At a flow point 510, the modified data blocks 200 and new data blocks
13 200 are held in the memory 103 and have not yet been written to disk.

14
15 Because no data block 200 is rewritten in place, each original control
16 block 200 which references a modified data block 200 (and each original control block
17 200 which references a modified control block 200) must be replaced with a modified
18 control block 200, all the way up the tree structure 400 to the root object 200.

19
20 At a step 521, for each modified data block 200, a free storage block on the
21 mass storage 104 is allocated for recording the modified data block 200. The blockmap
22 object 210 is altered to reflect the allocation of the storage block for the modified data
23 block 200 and freeing of the storage block for the original data block 200.

1 The blockmap object 210 maintains information about which storage
2 blocks on the mass storage 104 are allocated and have data stored therein, and which
3 storage blocks are free and eligible for use. The cache engine 100 searches the
4 blockmap object 210 for a free storage block, maintaining a write pointer 250 into the
5 blockmap object 210 so as to perform the search in a round-robin manner. Thus, when
6 the write pointer 250 advances past the end of the blockmap object 210, it is wrapped
7 around to the beginning of the blockmap object 210. The write pointer 250 is
8 maintained in the root object 220 so that the search continues in a round-robin manner
9 even after a failure and restart of the cache 102.

10
11 To maintain consistency of the cache 102 in the event of a failure, a free
12 storage block 200 cannot be considered free (and therefore used) if it is still referenced,
13 even if indirectly, by the root object 220. Accordingly, those blocks 200 which are
14 freed prior to atomic commitment of the root object 220 are not considered free until the
15 root object 220 is atomically written to disk.

16
17 At a step 522, for each original control block 200 which references an
18 original block 200 which is to be modified in this write episode, a modified control block
19 200 is generated. In like manner as the step 521, a free storage block on the mass
20 storage 104 is allocated for recording the modified control block 200. In like manner as
21 the step 521, the blockmap object 210 is modified to reflect the allocation of the storage
22 block for the modified control block 200 and freeing of the storage block for the original
23 control block 200.

1 The step 522 is repeated for each level of the tree structure 400 up to the
2 root object 220.

3
4 At a step 523, the operations of the step 521 and the step 522 are repeated
5 for those blocks 200 of the blockmap object 210 which were altered.

6
7 At a step 524, the modified data blocks 200 and modified control blocks
8 200 (including the blockmap object 210) are written to their allocated storage blocks on
9 the mass storage 104.

10
11 At a step 525, the root object 220 is rewritten in place on the mass storage
12 104.

13
14 At a flow point 530, the root object 220 has been rewritten in place, all
15 changes to the tree structure 400 have thus been atomically committed; the modified
16 blocks 200 have become part of the tree structure 400 and the original blocks 200
17 which were replaced with modified blocks 200 have become freed and eligible for reuse.
18 The modified blockmap object 210 is not atomically committed until the root object 220
19 has been rewritten in place, so storage blocks which are indicated as allocated or free are
20 not so indicated until the write episode has been atomically committed at the flow point
21 530.

22

1 When the modified blocks 200 are actually allocated to storage blocks and
2 written to those storage blocks on the mass storage 104, they are written in the
3 following manner:

4
5 o the tree structure 400 is traversed in a depth-first top-down manner, so as to
6 ensure that modified control blocks 200 are written in a sequence of storage
7 blocks before the modified data blocks 200 they reference;

8
9 o at each modified control block 200, the referenced modified data blocks 200 are
10 traversed in a depth-first top-down manner, so as to ensure that the referenced
11 modified data blocks 200 are clustered together in a sequence of storage blocks
12 after the modified control block 200 which references them.

13
14 This technique helps to ensure that when reading control blocks 200, the
15 data blocks 200 they reference are read-ahead whenever possible, so as to minimize the
16 number of operations required to read the control blocks 200 and the data blocks 200
17 from the mass storage 104.

18
19 The cache engine 100 determines when to perform a write episode, in
20 response to the condition of the memory 103 (including the number of modified blocks
21 200 in the memory 103), the condition of the mass storage 104 (including the number of
22 free storage blocks available on the mass storage 104), and the condition of the cache
23 102 (including the hit rate of network objects 114 in the cache 102).

1 In a preferred embodiment, write episodes using the method 500 are
2 performed upon either of the following conditions:

- 3
- 4 o when a certain time (such as 10 seconds) have elapsed since the previous write
5 episode; or
 - 6
 - 7 o when modified blocks comprise too large a proportion of memory.
 - 8

9 Write episodes using the method 500 can also be performed upon either of
10 the following conditions:

- 11
- 12 o the number of modified blocks 200 in the memory 103 is near the number of
13 available free storage blocks on the mass storage 104 minus the number of
14 storage blocks needed for the blockmap object 210; or
 - 15
 - 16 o the fraction of modified blocks 200 in the memory 103 is near the miss rate of
17 network objects 114 in the cache 102.
 - 18

19 However, the number of free blocks 200 on the mass storage 104 is
20 normally much larger than the number of blocks 200 to be written during the write
21 episode.

1 Each object 210 has an associated "access time," which indicates when
2 that object 210 was last written or read. However, it is not desirable to update the
3 access time on disk for each object 210 whenever that object 210 is read, as this would
4 produce a set of modified control blocks 200 (which must be written to disk during the
5 next write episode) whenever any object 210 is read.

6
7 Accordingly, a volatile information table is maintained which records
8 volatile information about objects 210, including access times for objects 210 which
9 have been read, and number of accesses for those objects 210. When an object 210 is
10 read, its access time is updated only in the volatile information table, rather than in the
11 object descriptor 211 for the object 210 itself. The volatile information table is
12 maintained in the memory 103 and is not written to disk.

13
14 In a preferred embodiment, network objects 114 can continue to be read
15 while write episodes using the method 500 are being performed, even for those network
16 objects 114 which include modified data blocks 200, because the modified data blocks
17 200 continue to be maintained in the memory 103 while the write episodes are
18 performed, whether or not they are actually successfully written to the mass storage 104.

19 20 *Removing Objects from Cache*

21
22 Figure 6 shows a block diagram of a set of pointers and regions on mass
23 storage.

1 A set of storage blocks on each disk drive of the mass storage 104 is
2 represented by a circular map 600, having indexes from zero to a maximum value Nmax.
3 In the figure, indexes increase in a counterclockwise direction, wrapping around from
4 the end to the beginning of each disk drive modulo the maximum value Nmax.

5
6 A DT (delete table) object 210 is maintained which includes an entry for
7 each deletable object 210. Each time one of the hash buckets 340 in the hash table 350
8 is accessed, a reference is inserted into the DT object 210 for each object 210 which is
9 referenced by one of the hash entries 360 in that hash bucket 340 and which qualifies
10 as deletable.

11
12 In alternative embodiments, an objectmap object 210 is maintained which
13 includes an entry for each of the blockmap entries in the blockmap object 210. In such
14 alternatives, each entry in the objectmap object 210 is either empty, which indicates that
15 the corresponding block 200 does not comprise an object descriptor 211, or non-empty,
16 which indicates that the corresponding block 200 comprises an object descriptor 211,
17 and further includes information to determine whether the corresponding object 210 can
18 be deleted. Each non-empty entry in the objectmap object 210 includes at least a hit
19 rate, a load time, a time to live value and a hash signature 330 for indexing into the hash
20 table 350.

21
22 The cache engine 100 searches the blockmap object 210 for a deletable
23 object 210 (an object 210 referenced by the DT object 210), maintaining a delete pointer

1 260 into the blockmap object 210, similar to the write pointer 250, so as to perform the
2 search in a round-robin manner. Thus, similar to the write pointer 250, when the delete
3 pointer 260 advances past the end of the blockmap object 210, it is wrapped around to
4 the beginning of the blockmap object 210. Also similar to the write pointer 250, the
5 delete pointer 260 is maintained in the root object 220 so that the search continues in a
6 round-robin manner even after a failure and restart of the cache 102.

7
8 The write pointer 250 and the delete pointer 260 for each disk drive in the
9 mass storage 104 each comprise an index into the map 600.

10
11 In a preferred embodiment, the delete pointer 260 is maintained at least a
12 selected minimum distance d0 601 ahead of the write pointer 250, but not so far ahead
13 as to wrap around again past the write pointer 250, so as to select a delete region 610 of
14 each disk drive for deleting deletable objects 210 which is near to a write region 620
15 used for writing modified and new objects 210. The write region 620 is at least the size
16 specified by the minimum distance d0 601. Although there is no specific requirement for
17 a size of the delete region 610, it is preferred that the delete region 610 is several times
18 (preferably about five times) the size of the write region 620. The cache engine 100 thus
19 provides that nearly all writing to disk occurs in a relatively small part of each disk drive.
20 This allows faster operation of the mass storage 104 because a set of disk heads for the
21 mass storage 104 must move only relatively a small distance during each write episode.

22

1 Because the cache engine 100 attempts to maintain a relatively fixed
2 distance relationship between the write pointer 250 and the delete pointer 260, write
3 episodes and delete episodes will occur relatively frequently. In a preferred embodiment,
4 the cache engine 100 alternates between write episodes and delete episodes, so that
5 each delete episode operates to make space on disk for a later write episode (the next
6 succeeding write episode writes the blockmap object 210 to disk, showing the blocks
7 200 to be deleted; the write episode after that is able to use the newly free blocks 200)
8 and each write episode operates to consume free space on disk and require a later delete
9 episode.

10 A collection region 630 is selected near to and ahead of the delete region
11 610, so as to select objects 210 for deletion. A size of the collection region 630 is
12 selected so that, in an time estimated for the write pointer 250 to progress through the
13 collection region 630 (this should take several write episodes), nearly all hash entries
14 360 will have been accessed through normal operation of the cache engine 100. Thus,
15 because each hash entry 360 includes information sufficient to determine whether its
16 associated object 210 is deletable, nearly all objects 210 will be assessed for deletion in
17 the several write episodes needed for the write region 620 to move through the
18 collection region 630.

19
20
21 Objects 210 which have been assessed for deletion are placed on an
22 deletion list, sorted according to eligibility for deletion. In a preferred embodiment,
23 objects 210 are assessed for deletion according to one of these criteria:

1 o If an object 210 is explicitly selected for deletion by the cache engine 100 due to
2 operation of the HTTP protocol (or a variant thereof, such as SHTTP), the object
3 210 is immediately placed at the head of the deletion list.

4
5 o If a new object 210 with the same name is created, the old object 210 is placed at
6 the head of the deletion list as soon as all references to the old object 210 are
7 released (that is, no processes on the cache engine 100 reference the old object
8 210 any longer).

9
10 o If an object 210 has expired, it is immediately placed at the head of the deletion
11 list.

12
13 o If a first object 210 has an older access time than a second object 210, the first
14 object 210 is selected as more eligible for deletion than the second object 210,
15 and is thus sorted into the deletion list ahead of the second object 210.
16

17 A fraction of objects 210 on the deletion list chosen due to the last two of
18 these criteria (that is, due to expiration or older access time), preferably one-third of the
19 objects 210 on the deletion list, are selected for deletion.

20

21 After each write episode, the collection region 630 is advanced by an
22 expected size of the next write region 620. In a preferred embodiment, the expected
23 size of the next write region 620 is estimated by averaging the size of the write region

1 620 for the past several (preferably seven) write episodes. Those objects 210 which
2 were on the deletion list before advancing the delete region 610 and which are in the
3 delete region 610 afterward are scheduled for deletion; these objects are selected
4 individually and deleted in the next delete episode (in a preferred embodiment, the next
5 delete episode is immediately after completion of the write episode).

6
7 In a preferred embodiment, write episodes and delete episodes for each
8 disk drive on the mass storage 104 are independent, so there are separate deletion
9 regions 610, write regions 620, and collection regions 630 for each disk drive on the
10 mass storage 104.

11 12 *Alternative Embodiments*

13
14 Although preferred embodiments are disclosed herein, many variations are
15 possible which remain within the concept, scope, and spirit of the invention, and these
16 variations would become clear to those skilled in the art after perusal of this application.